## 2.2 Flow Component Classes

### 2.2.1 EventCatcher Class

The EventCatcher class, which implements the AWT Event Listener interface responds to various events that the User and the Player initiate. These events could be a click on the screen, a click on a button, or a key press on the keyboard. The Swing toolkit is used to handle these possible events, which is located in the Javax package.

The EventCatcher class contains the eventDispatched(AWTEvent) method, which is the hub of all events and it is located in the main EventCatcher class. It calls the delegate methods to perform the specific job based on the particular event. This method accepts the AWTEvent as a parameter.

### 2.2.2 Settings Class

The Settings class stores the information that the user would like to store about the future match. For example, the volume of the background sound during the match, whether or not the logs should be stored to the disk and additional settings such as keyboard inputs that the user would like to use for movement and direction of the pieces.

The Settings class contains the following fields:

* Int soundVolume: the loudness of the background sound on a scale from 1 to 100.
* Boolean saveLogsToDisk: should the logs be stored to the disk for future reference?
* Dictionary<String, Integer> keyMapping: keyMapping stores the mapping of the keyboard key inputs to their functionalities.

The methods in the Settings class are the accessor methods and the mutator methods for the fields.

### 2.2.3 PlayerSettings Class

The PlayerSettings class contains information about the preferences of the player while playing the game. This class contains the following fields:

* Boolean isAI: isAI is a Boolean which denotes if the player is an AI or a human player.
* String name: name given to the player.
* Dictionary<String, RobotRecord> mapRobots: mapRobots is a dictionary which maps the name of the robot to the record of the robot.

The methods in this class are the accessor and mutator methods for each field. As an alternative to the setter for the dictionary, mapRobots, the method setProgram(String , RobotRecord) sets the AI piece identified by ID to a RobotRecord.

### 2.2.4 MatchOptions Class

The MatchOptions class is essentially for storing all the details about a match, if in case the player wants to rematch. The fields in this class include:

* PlayerSettings[] players: players is an array (maximum six) elements which stores the information about the player provided in the PlayerSettings class.
* Int numPlayers: numPlayers stores the number of players in the match.
* String mapName: mapName is the name of the map assigned to the match.
* Int numScouts, numSnipers, numTanks: As the name suggests, these variables store the number of scouts, snipers and Tanks of all players in the match.
* RobotRecord defaultScout, defaultSniper, defaultTank: These values store the record of the default Scout, Sniper and Tank, if the user chooses not to specify the Robot.

The methods in this class are the accessor and mutator methods for each of the fields.

### 2.2.5 RobotRecord Class

The RobotRecord class has the sole purpose of storing the JSON file.

The class contains only one field:

* String [] JSONFile: stores the JSON input as String in the form of an array.

### 2.2.6 MenuManager Class

The MenuManager class helps with the user inputs given on all screens that are interacted by the user. In other words, all events in all the screens except for the game screen are reported to the MenuManager class. As a result, the MenuManager class stores information about the match options and the general settings.

The class contains the following fields:

* Settings settings: this field stores the preferences from the settings
* MatchOptions matchOptions: matchOptions is the field which contains all the information about the match given in the MatchOptions class.
* List<RobotRecord> robotRecords: Stores a list of all records of the robots in the game.
* String [] buttons: stores all the possible button names in their scope of screens.

The method clickedButton(String ) is located both in MenuManager class and GameManager class. The method in the MenuManager deals with the buttons clicked on all screens other than the game screen, and it takes in the name of the button which is clicked as a parameter, throws an exception if the button name is invalid, calls the Display component to switch between the screens as requested or executes the expected request and returns nothing.

A private method IsValidButton() is used to check whether or not any of the arguments of button names are valid. The method loadSettings(), as the name suggests, loads settings from the file, or creates a new file if none exists. The method neither returns anything nor accepts any parameters. Similarly, the method saveSettings() updates the values of the settings and saves to file. Likewise, the method neither returns anything nor accepts any parameters.

### 2.2.7 GameManager Class

The GameManager class handles all the interactions between the player and the screen.

It consists of the following fields:

* MatchOptions m: stores the information about the match.
* String[] buttons: Like the MenuManager class, the GameManager class also contains a string of all possible button names in the game screen.

The class contains a method, createMatch(MatchOptions), which assigns the match options to the match field.

The method clickedButton(String ) is also located in the GameManager class. This method in the GameManager handles the buttons on the game screen. It takes in the name of the button which is clicked as a parameter, throws an exception if the button name is invalid, and returns nothing. The method clickedEndTurn() is called when the Player decides to end their turn. The method neither accepts any parameters nor returns a value. The draggedGameBoard( mousePosition pre, mousePosition post ) method is called when the player is panning the game board. This is a void-returning method and it accepts initial and final positions of the mouse. The method clickedMove(int ) is called when the Player decides to move their piece to a position. This void-returning method accepts an integer variable which refers to the position at which the piece wants to move. The method clickedShoot(int ) is called when the Player decides to shoot at a piece at a position. The method accepts an integer variable which refers to the position at which the piece wants to shoot and does not return any value. All of these methods report the change to the Board Component, and call the methodss accordingly.

The void-returning method clickedExitScreen() is called when the Player wants to quit the match. This method does not accept any parameters and reaches out to both the Display component to change the screen and the Board component indicating that the player wishes to quit the game. The method clickedPanArrow() is called when the Player chooses to switch between the Pan mode and the Arrow mode. This method contacts the Display component to recalculate the board interface on the screen.